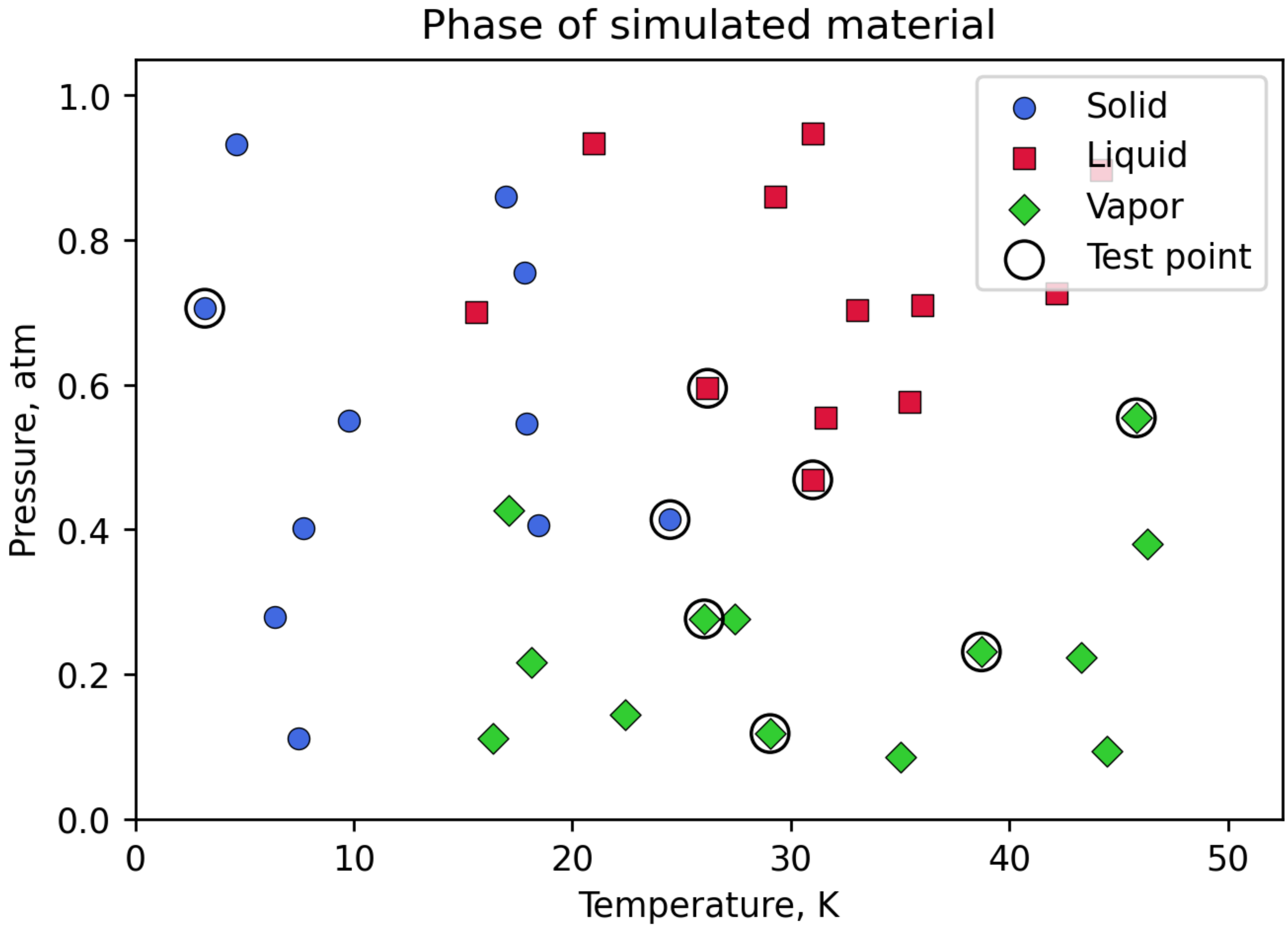
M8-L2 Problem 2

Let's revisit the material phase prediction problem once again. You will use this problem to try multi-class classification in PyTorch. You will have to write code for a classification network and for training.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import torch
from torch import nn, optim
import torch.nn.functional as F

def plot_loss(train_loss, val_loss):
    plt.figure(figsize=(4,2),dpi=250)
    plt.plot(train_loss,label="Training")
    plt.plot(val_loss,label="Validation",linewidth=1)
    plt.legend()
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.show()

def split_data(X, Y):
    np.random.seed(100)
    N = len(Y)
    train_mask = np.zeros(N, dtype=np.bool_)
    train_mask[np.random.permutation(N)[:int(N*0.8)]] = True
    train_x, val_x = torch.Tensor(X[train_mask,:]), torch.Tensor(X[np.logical_not(train_mask),:])
    train_y, val_y = torch.Tensor(Y[train_mask]), torch.Tensor(Y[np.logical_not(train_mask)])
    return train_x, val_x, train_y, val_y

x1 = np.array([7.4881350392732475,16.351893663724194,22.427633760716436,29.04883182996897,35.03654799338904,44.45894113
x2 = np.array([0.11120957227224215,0.1116933996874757,0.14437480785146242,0.11818202991034835,0.0859507900573786,0.0937
X = np.vstack([x1,x2]).T
y = np.array([0,2,2,2,2,2,0,2,2,2,2,2,0,0,2,0,1,2,0,0,1,1,1,2,0,1,0,1,1,1,0,0,1,1,1,1,])

X = torch.Tensor(X)
Y = torch.tensor(y,dtype=torch.long)

train_x, val_x, train_y, val_y = split_data(X,Y)
```

```python
def plot_data(newfig=True):
    xlim = [0,52.5]
    ylim = [0,1.05]
    markers = [dict(marker="o", color="royalblue"), dict(marker="s", color="crimson"), dict(marker="D", color="limegree
    labels = ["Solid", "Liquid", "Vapor"]

    if newfig:
        plt.figure(figsize=(6,4),dpi=250)

    x = X.detach().numpy()
    y = Y.detach().numpy().flatten()

    for i in range(1+max(y)):
        plt.scatter(x[y==i,0], x[y==i,1], s=40, **(markers[i]), edgecolor="black", linewidths=0.4,label=labels[i])

    plt.scatter(val_x[:,0], val_x[:,1],s=120,c="None",marker="o",edgecolors="black",label="Test point")

    plt.title("Phase of simulated material")
    plt.legend(loc="upper right")
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.xlabel("Temperature, K")
    plt.ylabel("Pressure, atm")
    plt.box(True)

def plot_model(model, res=200):
    xlim = [0,52.5]
    ylim = [0,1.05]
    xvals = np.linspace(*xlim,res)
    yvals = np.linspace(*ylim,res)
    x,y = np.meshgrid(xvals,yvals)
    XY = np.concatenate((x.reshape(-1,1),y.reshape(-1,1)),axis=1)
    XY = torch.Tensor(XY)
    color = model.predict(XY).reshape(res,res).detach().numpy()
    cmap = ListedColormap(["lightblue","lightcoral","palegreen"])
    plt.pcolor(x, y, color, shading="nearest", zorder=-1, cmap=cmap,vmin=0,vmax=2)
    return

plot_data()
plt.show()
```



Model definition

In the cell below, complete the definition for `PhaseNet`, a classification neural network.

- The network should take in 2 inputs and return 3 outputs.
- The network size and hidden layer activations are up to you.
- Make sure to use the proper activation function (for multi-class classification) at the final layer.
- The `predict()` method has been provided, to return the integer class value. You must finish `__init__()` and `forward()`.

In [2]:
```
class PhaseNet(nn.Module):
    def __init__(self):
        super().__init__()
        # YOUR CODE GOES HERE
        N_in = 2
        N_hidden = 12
        N_out = 3
        self.lin1 = nn.Linear(N_in,N_hidden)
        self.lin2 = nn.Linear(N_hidden,N_hidden)
        self.lin3 = nn.Linear(N_hidden,N_hidden)
        self.lin4 = nn.Linear(N_hidden,N_hidden)
        #self.lin5 = nn.Linear(N_hidden,N_hidden)
        self.lin5 = nn.Linear(N_hidden,N_out)
        self.act = F.relu

    def predict(self,X):
        Y = self(X)
        return torch.argmax(Y,dim=1)

    def forward(self,X):
        # YOUR CODE GOES HERE
        X = self.lin1(X)
        X = self.act(X)
        X = self.lin2(X)
        X = self.act(X)
        X = self.lin3(X)
        X = self.act(X)
        X = self.lin4(X)
        X = self.act(X)
        X = self.lin5(X)
        #X = self.act(X)
        #X = self.lin6(X)
        X = F.softmax(X, dim = 1)

        return X
```

Training

Most of the training code has been provided below. Please add the following where indicated:

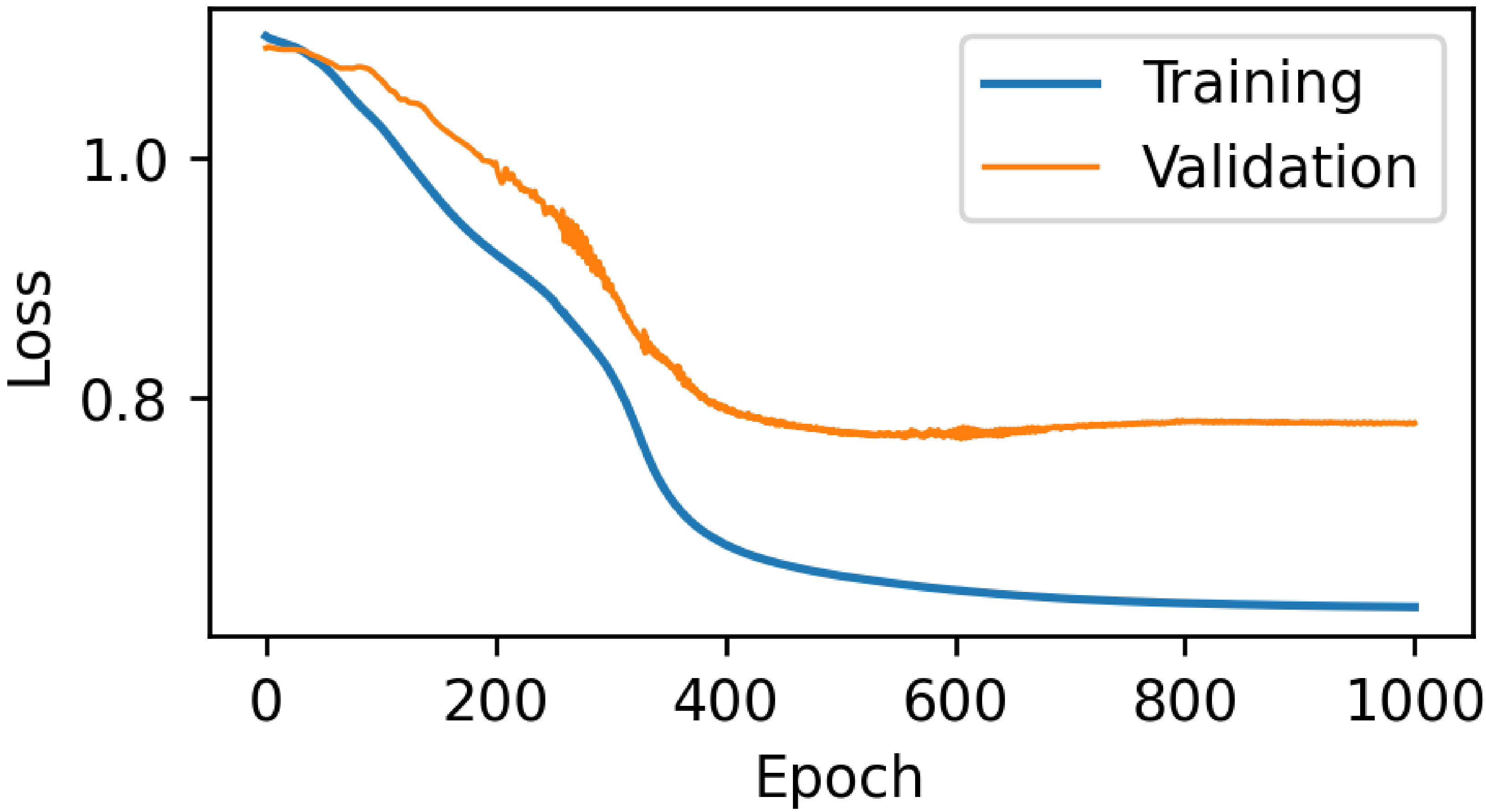
- Define a loss function (for multiclass classification)

- Define an optimizer and call it `opt`. You may choose which optimizer.

Make sure the training curves you get are reasonable.

In [3]:
```
model = PhaseNet()

lr = 0.001
epochs = 1000

# Define loss function
# YOUR CODE GOES HERE
lossfun = nn.CrossEntropyLoss()


# Define an optimizer, `opt`
# YOUR CODE GOES HERE
opt = optim.Adam(params = model.parameters(), lr=lr)

train_hist = []
val_hist = []

for epoch in range(epochs+1):
    model.train()
    loss_train = lossfun(model(train_x), train_y)
    train_hist.append(loss_train.item())

    model.eval()
    loss_val = lossfun(model(val_x), val_y)
    val_hist.append(loss_val.item())

    opt.zero_grad()
    loss_train.backward()
    opt.step()
    if epoch % int(epochs / 25) == 0:
        print(f"Epoch {epoch:>4} of {epochs}:   Train Loss = {loss_train.item():.4f}   Validation Loss = {loss_val.item


plot_loss(train_hist, val_hist)
```

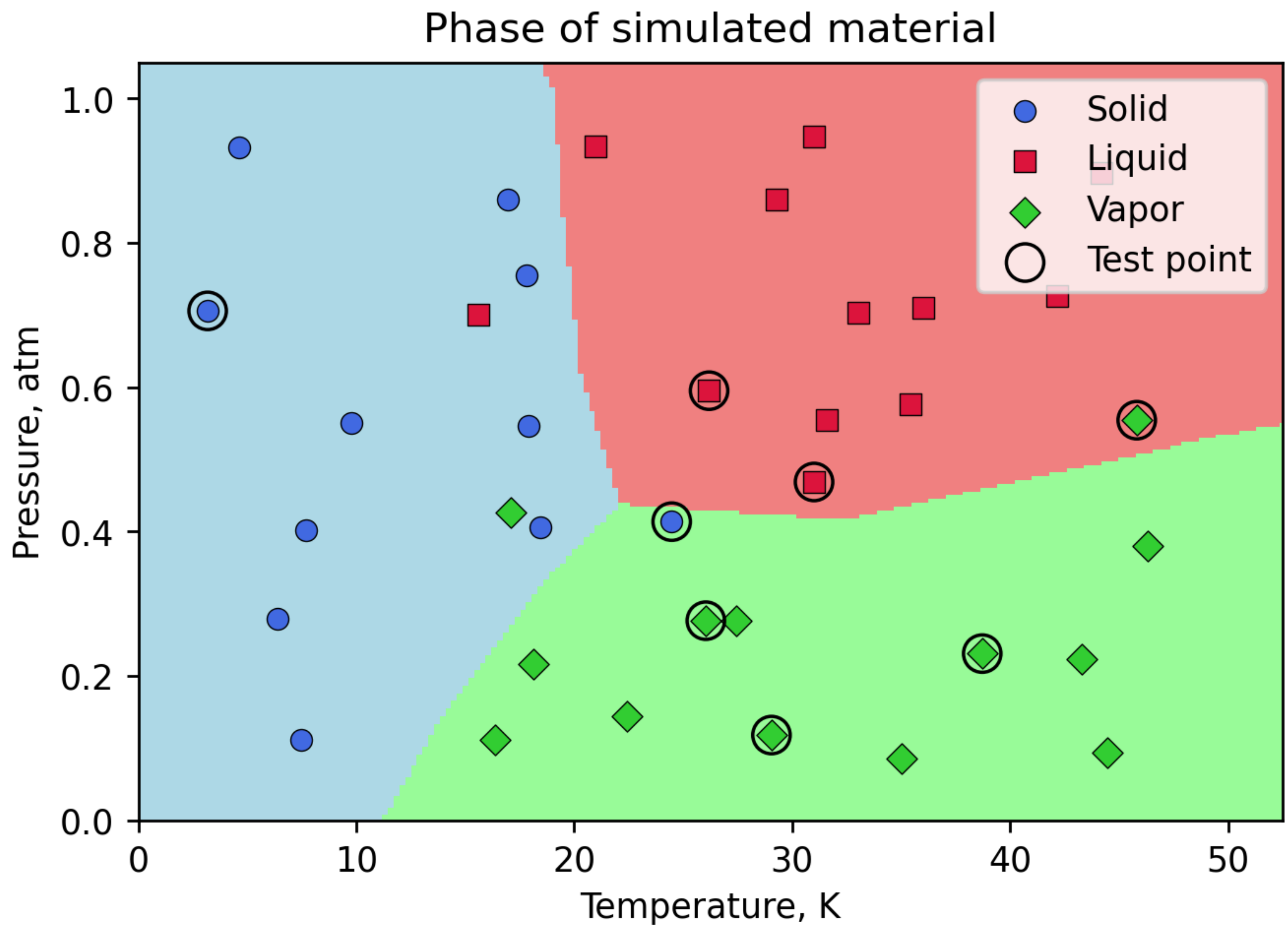
```
Epoch    0 of 1000:   Train Loss = 1.1018   Validation Loss = 1.0920
Epoch   40 of 1000:   Train Loss = 1.0839   Validation Loss = 1.0862
Epoch   80 of 1000:   Train Loss = 1.0454   Validation Loss = 1.0762
Epoch  120 of 1000:   Train Loss = 1.0017   Validation Loss = 1.0492
Epoch  160 of 1000:   Train Loss = 0.9548   Validation Loss = 1.0204
Epoch  200 of 1000:   Train Loss = 0.9200   Validation Loss = 0.9962
Epoch  240 of 1000:   Train Loss = 0.8903   Validation Loss = 0.9645
Epoch  280 of 1000:   Train Loss = 0.8470   Validation Loss = 0.9123
Epoch  320 of 1000:   Train Loss = 0.7806   Validation Loss = 0.8581
Epoch  360 of 1000:   Train Loss = 0.7065   Validation Loss = 0.8254
Epoch  400 of 1000:   Train Loss = 0.6772   Validation Loss = 0.7890
Epoch  440 of 1000:   Train Loss = 0.6631   Validation Loss = 0.7817
Epoch  480 of 1000:   Train Loss = 0.6544   Validation Loss = 0.7729
Epoch  520 of 1000:   Train Loss = 0.6482   Validation Loss = 0.7693
Epoch  560 of 1000:   Train Loss = 0.6433   Validation Loss = 0.7680
Epoch  600 of 1000:   Train Loss = 0.6390   Validation Loss = 0.7702
Epoch  640 of 1000:   Train Loss = 0.6357   Validation Loss = 0.7724
Epoch  680 of 1000:   Train Loss = 0.6330   Validation Loss = 0.7750
Epoch  720 of 1000:   Train Loss = 0.6310   Validation Loss = 0.7766
Epoch  760 of 1000:   Train Loss = 0.6294   Validation Loss = 0.7786
Epoch  800 of 1000:   Train Loss = 0.6281   Validation Loss = 0.7795
Epoch  840 of 1000:   Train Loss = 0.6272   Validation Loss = 0.7799
Epoch  880 of 1000:   Train Loss = 0.6265   Validation Loss = 0.7800
Epoch  920 of 1000:   Train Loss = 0.6259   Validation Loss = 0.7787
Epoch  960 of 1000:   Train Loss = 0.6255   Validation Loss = 0.7786
Epoch 1000 of 1000:   Train Loss = 0.6251   Validation Loss = 0.7786
```



Plot results

Plot your network predictions with the data by running the following cell. If your network has significant overfitting/underfitting, go back and retrain a new network with different layer sizes/activations.

In [4]:
```
plot_data(newfig=True)
plot_model(model)
plt.show()
```


Phase of simulated material
Solid
Liquid
Vapor
Test point
Pressure, atm
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Temperature, K

```
In [ ]:
```